Statistical Techniques in Robotics (16-831, F14) Lecture#23 (Tues, December 2)

How to apply Machine Learning

Lecturer: Drew Bagnell *Scribes: Jacob Walker, Vishnu Desaraju* [1]

This lecture provides some general guidelines that may be useful for application of various ML techniques in practice.

1. Your test data is sacred

   - Split data into training, validation (development) and test sets. Alternatively one can do k-fold cross validation. Cross-validation helps, but it's hard to tell exactly how much it helps. Be cautious of trajectory data (sequential information) or groupable data (partial exchangability, *e.g.*: different scenes or locations). In such cases it might make more sense to split based on trajectories or groups instead of individual points. A good rule of thumb is 80% training, 10% development, and 10% testing.

2. Understand your data

   It is a good idea to try and understand the characteristics of the data we are dealing with. Visualizing the data may reveal problems with the data itself, which otherwise may manifest themselves as hard to debug behavior of the algorithm using the data. Typical problems are

   (a) All the features are zero
   (b) Some feature has very large value due to noise or other reasons. Very large feature values may be problematic in certain algorithms e.g. SVMs.

   Visualizing the data may help us get an understanding of how it is distributed and will be useful for selecting an appropriate model. To visualize one can plot data. Visualization can reveal missing data, the separability of data, the distribution of classes, the relative scales of features, and outliers.

   - Histogram each feature
   - Scatter plot 2D/3D pairs or triplets of data
   - Perform PCA and scatter plot projected data
   - Random projections
   -

3. Don't use labels as features.

   - This seems blatantly obvious, but this error is dangerously easy to commit. Double check to make sure no information directly related to labels are in your feature set.

4. Normalize data

   - Mean centered and variance scaled.

[1] Some content adapted from previous scribes: Scott Satkin, Saurabh Singh

- Min-Max based scaling to $[-1, 1]$
- Whacky way. Project onto unit sphere, works well in case of SVM, $\frac{f}{||f||}$.
- Whiten the data. This is the **best** way to normalize data. Naive normalization is not rotationally invariant. Whitening will normalize along eigenvectors. One drawback is that whitening can amplify noise, thus it's advisable to combine whitening with PCA.

5. Cris Dima's rule:
   - "Always start by overfitting." (Get an upper bound on performance)
6. "More data is better". If you lost track of the conversation, you can always join back using this.
7. Understand what features matter
   - Throw in all features. Adding features should never hurt (in the limit of infinite data). Except -
     - Computational problems
     - Overfitting problems
     - Memory problems

     Ablative analysis
   - – Measure performances of all combinations of all but one feature on development set. Remove the least important one. Iterate. This is useful if you want largest number of features.
     - Add features one at a time: Start by measuring performance of all features individually. Include the most important one. Iterate. This is useful if you want smallest number of features.
       Since this may easily explode. Its better to follow a greedy strategy instead of exploring all combinations. "Greed is Good" - Tropp. Boosting is an example.
     - Analysis:
       * These work well when features are not too correlated.
       * Greedy: $F_1 G_\infty (\log d)\sqrt{T}$
         Not good for sparse features.
       * Alternative: $F_2 G_2 \sqrt{T}$
         $\sum |w_i| \leq F_1 \subset F_2$
8. Never underestimate the power of a linear predictor (Linear SVM, Logistic Regression, Linear Regression...).
9. Never underestimate the power of "well tuned" Gradient Descent. (Nelder-Mead if no derivatives).
10. Unbalanced classes Sometimes we may have disproportional data for various classes. Typical manifestation of this problem is classifier classifying everything to one class.

    Approaches to tackle this could be -

    (a) Downsample: Downsample the over-represented classes such that all classes have similar amount of data.

- Downside: doesn't use all the data

(b) Downsample ("costing"): Downsample multiple times and average results over all downsamplings.

- Downside: doesn't use all the data, still better than previous

(c) Upsample (duplicate data)

- Downsides: overfitting and computational overhead. In decision trees this might be a problem if there is a rule for splitting based on number of points.

(d) Weight the samples

i. Change margin (Margin Scaling): Doesn't change the gradient. Only affects the update.

ii. Scale loss (Scale slack)

$$\min ||w||^2 + \lambda \sum_{\alpha_i} \xi_i \tag{1}$$

Here λ can be thought of as weight for scaling the loss for some of the points.

11. Greedy Forward Selection

- Train classifier with all single features, pick the best feature (per unit time), add to final set, and repeat. ℓ_1 regularization or exponentiated are sophisticated approaches, but they are often no better than greed forward selection. Orthogonal matching pursuit is a fast approximation to greedy forward seleciton .

12. Understand Overfitting vs. Underfitting

- Overfitting

  You do great on training but relatively quite bad on testing.
  - Test on development set
  - Regularize more
  - Feature selection (less features). Step-wise, PCA, or just think hard
  - Get more data
  - Use simpler classifier
- Underfitting

  You don't perform quite good on both training and testing.
  - More features
  - More sophisticated classifier
  - Less regularization
  - Kernelize or boost
  - Optimize better

13. Premature Statistical Optimization

- Avoid that. Optimize bottleneck.

14. Don't buy the hype!

- Rich Caruana[2] papers
- SVM
- Random forests ("Random Kitchen Sinks")
- ANN (often 2nd best option)

15. Building Large Learning Systems

    - Premature statistical optimization (spend time on the parts that matter)
    - Unit test learning code

[2] http://www.cs.cornell.edu/~caruana/